

Copyright Warning & Restrictions

The copyright law of the United States (Title 17, United States Code) governs the making of photocopies or other reproductions of copyrighted material.

Under certain conditions specified in the law, libraries and archives are authorized to furnish a photocopy or other reproduction. One of these specified conditions is that the photocopy or reproduction is not to be “used for any purpose other than private study, scholarship, or research.” If a user makes a request for, or later uses, a photocopy or reproduction for purposes in excess of “fair use” that user may be liable for copyright infringement,

This institution reserves the right to refuse to accept a copying order if, in its judgment, fulfillment of the order would involve violation of copyright law.

Please Note: The author retains the copyright while the New Jersey Institute of Technology reserves the right to distribute this thesis or dissertation

Printing note: If you do not wish to print this page, then select “Pages from: first page # to: last page #” on the print dialog screen

The Van Houten library has removed some of the personal information and all signatures from the approval page and biographical sketches of theses and dissertations in order to protect the identity of NJIT graduates and faculty.

ABSTRACT

Object Oriented Design of Petri Net Simulator

by
Himanshu Juneja

Petri nets are highly useful for modeling discrete event dynamic systems. The objective of this effort is to develop a tool for drawing, editing and simulating Petri nets using object oriented programming on a standard platform. The object oriented approach was chosen because of its code reuse and extendability features which simplify the task of adding features to the tool as the model evolves. The design stresses on modeling of the problem by objects which closely relate the system design with the implementation.

C++ is used for implementing the object oriented design, and the XView toolkit is used for building the graphical editor in compliance with AT&T's OPENLOOK standards on a Sun Sparc IPX running SunOS 4.1.2. The object oriented paradigm was successfully applied to develop a user friendly, graphical editor and a simulator for Petri nets.

OBJECT ORIENTED DESIGN OF
PETRI NET SIMULATOR

by
Himanshu Juneja

A Thesis
Submitted to the Faculty of
New Jersey Institute of Technology
in Partial Fulfillment of the Requirements for the Degree of
Master of Science in Electrical Engineering

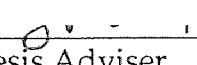
Department of Electrical and Computer Engineering

October 1993

APPROVAL PAGE

Object Oriented Design of
Petri Net Simulator


Himanshu Juneja



Dr Anthony D. Robbi, Thesis Adviser
Associate Professor of Electrical and Computer Engineering, NJIT

8/11/93

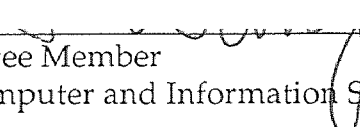
date



Dr Meng Chu Zhou, Committee Member
Assistant Professor of Electrical and Computer Engineering, NJIT

8/10/93

date



Dr David Wang, Committee Member
Assistant Professor of Computer and Information Science

August 10, 1993

date

BIOGRAPHICAL SKETCH

Author: Himanshu Juneja

Degree: Master of Science in Electrical Engineering

Date: October 1993

Date of Birth:

Place of Birth:

Undergraduate and Graduate Education:

- Master of Science in Electrical Engineering
New Jersey Institute of Technology, Newark, NJ, 1993
- Bachelor of Engineering in Electrical Engineering
Delhi Institute of Technology, Delhi, India, 1990

Major: Electrical Engineering

*This thesis is dedicated to
my brother Ashish Kapoor
and my father the late A.S. Juneja*

ACKNOWLEDGMENT

The author would like to express his sincere gratitude to his advisor, Professor Anthony D. Robbi, for his guidance, support, kindness and encouragement throughout this thesis.

Special thanks to Professor David Wang and Professor MengChu Zhou for serving as members of the thesis committee.

The author appreciates the help and suggestions from all the members of the Petri net group.

TABLE OF CONTENTS

| Chapter | Page |
|---|------|
| 1 INTRODUCTION | 1 |
| 1.1 Objectives | 1 |
| 1.2 Petri Nets Concepts | 2 |
| 1.2.1 Petri Net Terminology and Representation | 2 |
| 1.2.2 Petri Net Marking and Firing Rules | 3 |
| 1.2.3 Petri Net Liveness | 4 |
| 1.3 Petri Nets Applications | 4 |
| 1.4 Thesis Organization | 5 |
| 2 OBJECT ORIENTED CONCEPTS AND X WINDOW SYSTEM | 7 |
| 2.1 Basic Concepts of Object Oriented Programming | 7 |
| 2.1.1 Terminology | 7 |
| 2.1.2 Encapsulation or Data Abstraction | 8 |
| 2.1.3 Inheritance, Generalization and Specialization | 9 |
| 2.1.4 Polymorphism and Late Binding | 11 |
| 2.2 Reuse and Extensibility of Code | 11 |
| 2.3 X Window System Concepts | 12 |
| 2.3.1 Window Manager | 13 |
| 2.3.2 Writing X Applications and User Interface Standards | 13 |
| 2.3.3 XView Toolkit and Notifier based Approach | 13 |
| 3 DESIGN OF PETRI NET TOOL | 15 |
| 3.1 How OOP is Applied to Design of PNT | 15 |
| 3.2 The Controller Object | 16 |
| 3.3 The Net Object | 17 |
| 3.4 The Transition Object | 18 |
| 3.5 The Place Object | 19 |
| 3.6 The Arc Object | 20 |
| 3.7 The Segment Array Object | 20 |

| Chapter | Page |
|---|------|
| 3.2 The Segment Object..... | 21 |
| 3.3 The Basic_Object..... | 21 |
| 4 PNT USERS MANUAL..... | 23 |
| 4.1 Overview of PNT..... | 23 |
| 4.2 The File Menu..... | 24 |
| 4.2.1 The New Button..... | 25 |
| 4.2.2 The Load Button..... | 25 |
| 4.2.3 The Save Button..... | 25 |
| 4.2.4 The Quit Button..... | 26 |
| 4.3 The Draw Menu..... | 26 |
| 4.3.1 The Place Button..... | 27 |
| 4.3.2 The Horizontal and Vertical Transition Buttons..... | 27 |
| 4.3.3 The Normal and Inhibit Arc Buttons..... | 27 |
| 4.3.4 The Token Button..... | 28 |
| 4.4 The Edit Menu..... | 28 |
| 4.4.1 The Tag and Arctag Button..... | 29 |
| 4.4.2 The Eraser and Erase Text Buttons..... | 31 |
| 4.4.3 The Clear and KillText Buttons..... | 31 |
| 4.4.4 The Delseg and Delarc Buttons..... | 31 |
| 4.5 The Simulate Menu..... | 32 |
| 4.6 The Utilities Menu..... | 33 |
| 4.8 Control Flow in PNT..... | 35 |
| 5 CONCLUSION..... | 37 |
| 5.1 Enhancement and Portability issues..... | 37 |
| 5.2 Summary..... | 38 |
| APPENDIX..... | 39 |
| REFERENCES..... | 46 |

LIST OF TABLES

| Table | Page |
|---|------|
| 1.1 Typical Interpretations of Transitions and Places | 5 |
| 3.1 Attributes and Operations of Controller Object. | 17 |
| 3.2 Attributes and Operations of Net Object. | 18 |
| 3.3 Attributes and Operations of Transition Object | 19 |
| 3.4 Attributes and Operations of Place Object. | 19 |
| 3.5 Attributes and Operations of Arc Object. | 20 |
| 3.6 Attributes and Operations of Segment Array Object. | 21 |
| 3.7 Attributes and Operations of Segment Object. | 21 |
| 3.8 Attributes and Operations of Basic Object. | 22 |

LIST OF FIGURES

| Figure | Page |
|--|------|
| 1.1 A Simple Petri Net | 2 |
| 2.1 Real World Object Hierarchy..... | 8 |
| 2.2 XView Object Hierarchy | 10 |
| 2.3 Software Architecture of PNT..... | 14 |
| 4.1 PNT in Openwindows Canvas | 23 |
| 4.2 The File Menu in Openwindows Canvas | 25 |
| 4.3 The Draw Menu in Openwindows Canvas..... | 26 |
| 4.4 The Edit Menu in Openwindows Canvas..... | 29 |
| 4.5 Tag Panel for Place..... | 30 |
| 4.6 Tag Panel for Transition..... | 30 |
| 4.7 Tag Panel for Arc..... | 30 |
| 4.8 The Simulate Menu in Openwindows Canvas | 32 |
| 4.9 Panel for Simulation in Openwindows Canvas..... | 32 |
| 4.10 The Utilities Menu in Openwindows Canvas..... | 33 |
| 4.11 A Sample Verify Window in Openwindows Canvas..... | 34 |
| 4.12 A Sample Log Window in Openwindows Canvas..... | 35 |

CHAPTER 1

INTRODUCTION

1.1 Objectives

The objective of this effort is to build a user friendly, interactive graphical simulation tool for Petri nets using object oriented programming. The tool has capabilities to draw, edit and simulate a Petri net which models any discrete event dynamic system examples include communication protocol, flexible manufacturing system and system software. Once such system is translated into a Petri net model, the model is useful for studying system performance and behavior. At present an object oriented computer tool with this capability does not exist. Earlier Petri net tools have been developed by Chiola[2] and Feldbrugge[5]. A suite of tools in C language have been developed by graduate students at NJIT[3], [11], [12]. They lack the extensibility of the object oriented style, do not support X Windows and are not fully operational.

The taxonomy of the Petri nets is so wide that implementations for all kinds of Petri nets by one student is impossible. The object oriented approach was chosen because of its properties like inheritance and generalization which make code reusability much easier. This approach relates the design phase very closely to the implementation phase which eases debugging and management. The current tool has used an entirely new design for the simulator. Some of the features of the previous user interface have been retained but the implementation is in compliance with OPENLOOK standards of graphical user interface (GUI) design. It is in a networkable windowing system unlike the previous tools which used a kernel-tied windowing system. The biggest challenge in using this approach is software design which is much closely related to the implementation phase than the modular approach. Considerable effort has been made in the design of the tool and the selection and definition of various objects to make it provide a robust infrastruc-

ture for adding new features and porting it to other platforms.

1.2 Petri Net Concepts

A Petri net is a bipartite graph for modeling discrete event dynamic systems. Petri net theory was developed by Carl Adam Petri in 1962[9]. A Petri net are abstract and formal model of information flow[8]. The properties, concepts and techniques of Petri nets have been developed to obtain natural, simple and powerful methods for describing and analyzing the flow of control in systems, particularly those which exhibit asynchronous and concurrent activities. As a graphical tool Petri nets can be used as a visual communication aid similar to flowcharts, block diagram and networks with much stronger modeling power. Analysis of a Petri net can reveal important information about the dynamic behavior of the modeled system. This information can then be used to suggest improvements or changes in the system and its parameters.

1.2.1 Petri Net Terminology and Representation

In a Petri net the graph nodes are the places, transition and tokens. In a Petri net graph, a place is represented by a circle and a transition is represented by a solid bar. The places and the transitions are connected by directed lines called arcs.

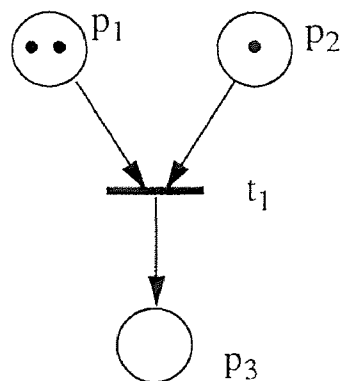


Figure 1.1 A Simple Petri Net

If p_i is an input place of a transition t_j , the connecting arc is pointed to transition.

If p_i is an output place of a transition t_j , then the connecting arc is pointed away from the transition. For example if p_1 and p_2 are input places of t_1 and p_3 is an output place of t_1 , in Fig 1.1.

The interpretation of the places and transitions is related to the modeled system. For example in Fig 1.1 p_1 could represent the availability of a part in a factory cell, p_2 could represent availability of robot, t_1 could represent the operation of robot moving the part in the destination, and p_3 could represent the destination or a buffer. The tokens in this place indicate availability. The presence of the required part and robot leads to the operation. Transition t_1 fires removing the tokens from p_1 and p_2 and depositing a token in p_3 . Thus a marked net represents the state of the system and the movement of tokens describes the dynamics of the system. Petri nets provide a natural representation of the systems where control and state information is distributed. The use of finite state machine to model these problems could lead to an unmanageably large number of states.

Mathematically a Petri net is composed of four parts: a finite set of places P , $P = \{p_1, p_2, \dots, p_n\}$, where $n \geq 0$; a finite set of transitions T , $T = \{t_1, t_2, \dots, t_m\}$, where $m \geq 0$; an input function I , and an output function O . The set of places and the set of transitions are disjoint, $P \cap T = \emptyset$.

$I: P \times T \rightarrow \mathbb{N}$ where $\mathbb{N} = \{0, 1, 2, \dots\}$ is the input function, for all $p \in P$ and all $t \in T$ such that $I(p, t)$ is a non negative integer. $O: P \times T \rightarrow \mathbb{N}$ where $\mathbb{N} = \{0, 1, 2, \dots\}$ is the output function, for all $p \in P$ and all $t \in T$ such that $O(p, t)$ is a non negative integer.

1.2.2 Petri Net Marking and Firing Rules

A marking m is an assignment of tokens to the places of a Petri net. Tokens reside in the places in a Petri net. Pictorially tokens are represented by dots or by an integer. The number and position of tokens normally change during the execution of a Petri net.

A transition $t_j \in T$ in a marked Petri net with marking m is enabled if each of its input places has at least as many tokens in it as arcs from the places to the transition. For drawing convenience, multiple parallel arcs are represented by a single arc with an integer weight. That is, for all $p \in P$,

$$\mu(p) \geq I(p,t)$$

A transition can fire if it is enabled. Firing a transition will in general change the marking μ of the Petri net to a new marking μ' .

A transition fires by removing an input token per arc from its input places and then depositing into each of its output places one token for each arc from the transition to the place. That is, for all $p \in P$,

$$\mu'(p) = \mu(p) - I(p,t) + O(p,t)$$

1.2.3 Petri Net Liveness

A deadlock in a Petri net is a marking in which no transition can fire. The execution of a Petri net is the firing of all enabled transitions in one state (marking) to reach another state. A live Petri net guarantees deadlock-free operation no matter what firing sequence is chosen. If a Petri net represents a working system, then it should be kept live.

1.3 Petri Net Applications

Conventional systems are unable to model concurrent systems successfully. On the other hand Petri nets have demonstrated a powerful approach towards the solution for concurrency and parallelism problems. The knowledge of the fundamentals of Petri net theory is becoming mandatory for various engineering disciplines. Due to the generality and permissiveness inherent in Petri nets, they have been proposed for a very wide variety of applications like modeling and analysis of distributed-software systems, distributed-database systems, concurrent and parallel programs, flexible manufacturing/industrial control systems, multiprocessor memory systems, dataflow computing systems, fault-

tolerant systems, programmable logic and VLSI arrays, asynchronous circuits and structures, compiler and operating systems, office-information systems, formal languages, and logic programs.

In Petri net modeling, using the concept of conditions and events, places represent conditions, and transitions represent events. A transition has a certain number of input and output places. The input places of a transition represent the preconditions of the corresponding event and the output places the postconditions. The concurrence of events corresponds to the simultaneous firing of the corresponding transitions.

The presence of tokens in a place is interpreted as holding the truth of the condition associated with the place. In another interpretation, k tokens are put in a place to indicate that k data items or resources are available. When a transition fires it removes the tokens representing the truth of the precondition and creates new tokens which represent the truth of postconditions. Some typical interpretations of transitions and their input places and output places are shown in Table 1.1

Table 1.1 Typical Interpretations of Transitions and Places

| Input Places | Transitions | Output Places |
|------------------|------------------|------------------|
| Precondition | Event | Postconditions |
| Input data | Computation step | Output data |
| Input signals | Signal processor | Output signals |
| Resources needed | Task or job | Resource release |
| Conditions | Clause in logic | Conclusion(s) |
| Buffers | Processor | Buffers |

1.5 Thesis Organization

The next chapter introduces the reader to object oriented programming, the basic concepts and features which make it so attractive. A comparison is drawn

between object oriented and conventional modular programming. Then it provides the basics of the X Window System which is a network based windowing system employing client-server design.

The third chapter provides an in depth discussion of the software design and how the object oriented programming paradigm was applied to achieve the desired design. All the objects defined are discussed in detail including their attributes, operations and how they communicate with other objects. Finally the integration with XView (an Object Oriented Toolkit) to put the complete tool together is described.

Chapter Four is the User's Manual of PNT(Petri Net Tool). All the capabilities of the tool are described in detail, how to use the tool to draw and edit a Petri net using the graphical user interface and how to simulate its execution. It describes all the buttons and their usage and various associated files. The conclusion of this thesis includes a discussion of portability and enhancement issues.

CHAPTER 2

OBJECT ORIENTED CONCEPTS AND X WINDOW SYSTEM

2.1 Basic Concepts of Object Oriented Programming

An object is an abstraction of an identity capable of independent existence, something which has properties and is not just a property itself. For example a thermometer is an object but temperature is not, because temperature is not capable of independent existence. A fundamental concept behind object oriented programming which separates it from modular programming, is the binding of data with its associated functionality, model the problem in terms of real world objects with data and functions associated with them. C++ provides this by allowing structures to include function definitions. Although a more general data type, class is also provided. In theory the modularity achieved by “top-down” design ought to provide software components that fit well together. In practice the fit is seldom perfect, and conventional software components nearly always require modification before they can be reused. Object oriented programming changes all this, by providing built-in techniques for managing the software development.

2.1.1 Terminology

A “class” is a group of objects with similar properties (attributes) and common behavior (operations). “Attributes” are data values held by an object or simply the properties of an object. Operations are the behavior of an object in response to a stimulus or a message from another object. The implementation of operations for a class is called “method”. As an example from the real world, take the example of motor vehicles which is a class of similar objects such as cars, motorcycles etc, see Fig 2.1. They all have certain properties like power, engine volume, fuel consumption, etc, which are the attributes of the class but the values for each instance of the class may not be the same. Similarly each instance of the class is

capable of some operations like driving which describe the behavior of the object. Since driving a car would be different than driving a motorcycle (though functionally similar) the specific implementation is called method. The above concept is used in modeling a problem in terms of objects. Their class membership, their attributes and their operations are defined. The choice of objects depends on the problem at hand. The figure 2.1 illustrates the above discussed concept.

2.1.2 Encapsulation or Data Abstraction

If we keep in mind a metaphor that reflects the way real world object exist and interact, we can create “software objects” that exhibit attributes and behaviors

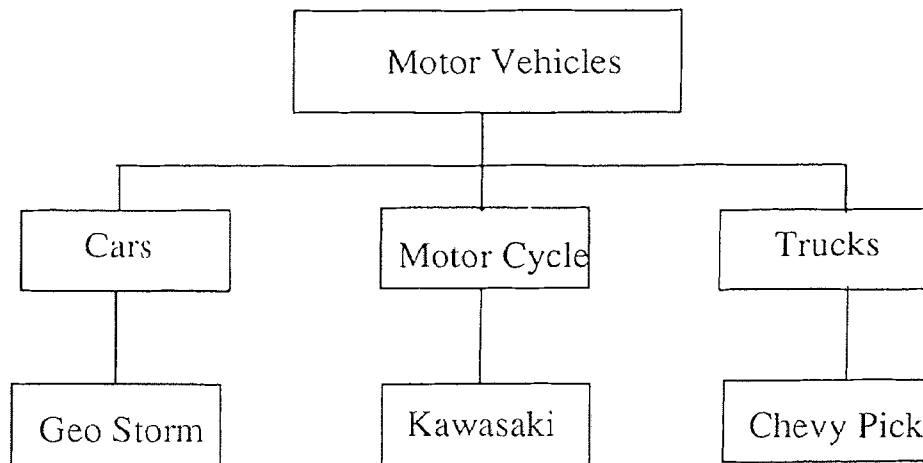


Figure 2.1 A Real World Object Hierarchy

“Encapsulation” defines a data structure of attributes and a group of member function as a single unit called an object. Object attributes in C++ are stored in data structures that resemble ordinary C structures. Object behaviors are implemented as functions called member functions in C++.

C++ affords the programmer a great deal of flexibility in controlling access to an object’s attributes and member functions. For example attributes and member functions declared to be “private” cannot be accessed from outside the object, except by functions declared to be “friends” of the object. Attributes and member

functions declared to be *public* can be accessed by any object, while those declared to be *protected* can only be accessed by certain objects. This OOP feature is called data hiding and it limits the visibility of data and allows it to be manipulated only via public member functions.

Data hiding enhances reliability and modifiability of software by reducing the interdependencies between the objects. In fact, if public member functions are specified correctly, the private data structures and member functions of an object may be changed without affecting the way other objects are implemented. This hiding of data is analogous to our experience in the real world, where there is often no need to know, how the internals of an object, such as a telephone, work.

2.1.3 Inheritance, Generalization and Specialization

Perhaps the most powerful feature of object oriented programming is "Inheritance", which allows objects to acquire the attributes and behavior of other objects. Inheritance contributes to economical and maintainable design, because objects can share attributes and behaviors without duplicating the program code that implements them.

Classes with similar attributes and operations may be organized into a hierarchial relationship. All common attributes can be factored out and assigned to a broader SuperClass, this is also referred to as "Generalization". A class can be iteratively redefined into SubClasses that inherit the attributes and operations of SuperClass. This is called "Specialization". Generalization is transitive in the sense that each instance of the subclass is an instance of its ancestor class and each descendent class refines its ancestor by adding new attributes and operations (though there is no strict requirement to refine for inheritance). Each object has a value for every attribute in its chain of ancestors and each operation available to its ancestor is available to it. A very good analogy from real world is the taxonomic scheme used by zoologists and botanists to classify living things

Software objects occupy a hierarchy in much the same way as real world

objects. A very good example is the object oriented toolkit XView, which is used for the user interface design of the PNT. This toolkit has various objects like Menu, Font, Window, Cursor, Screen, Frame, Server etc. These objects are implemented in a certain hierarchy, see Fig 2.2 for example Frame is a SubClass of Window, as it has all the properties of Window, but also has several properties of its own such as a header, footer etc. The hierarchy has multiple level, as there is SuperClass

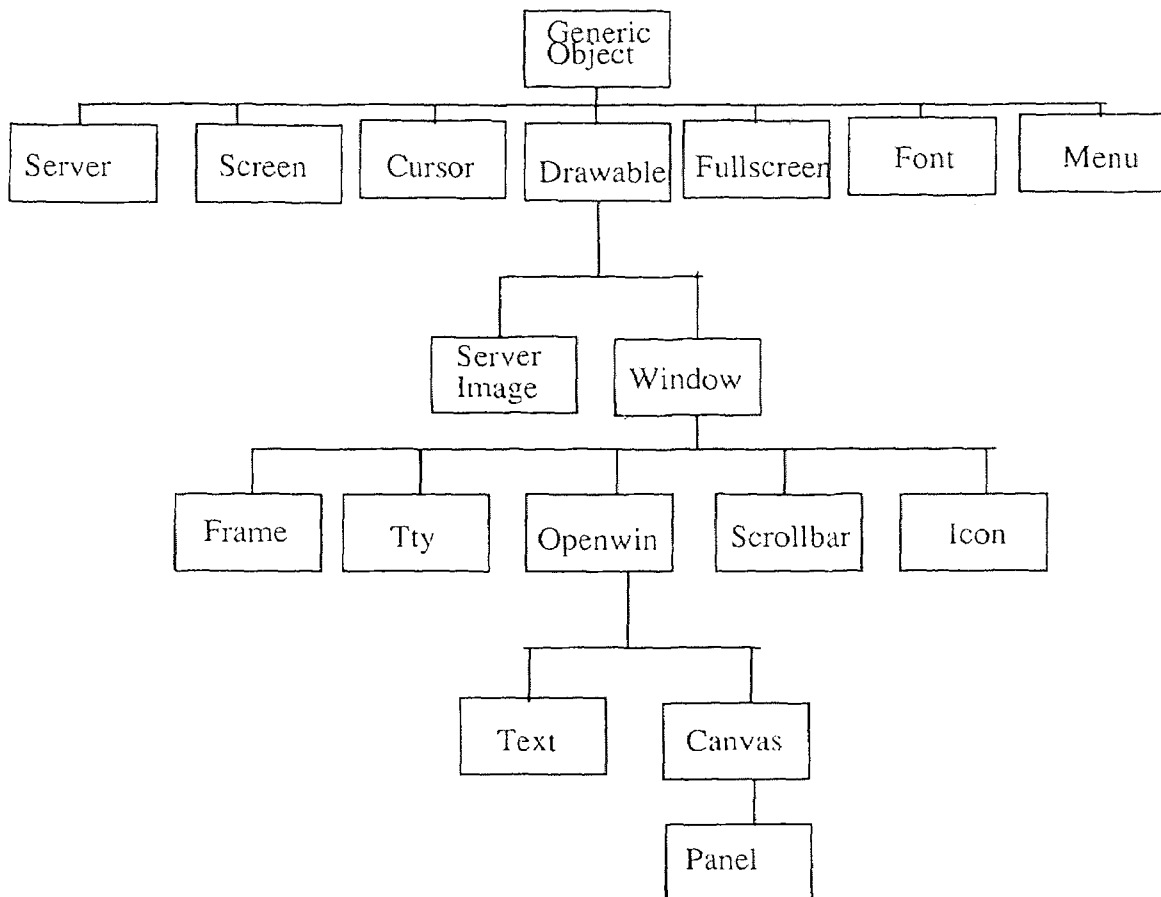


Figure 2.2 XView Object Hierarchy

called "Generic Object" from which all the objects are defined. To illustrate the benefit of inheritance consider the following argument. When the Window Class is created, a move operation is defined for moving the Window in the workspace. Frame inherits that function from Window, which means Frames can be moved

around without reimplementing the code and it helps guarantee that if the move function is implemented correctly for class Window it is correct throughout the hierarchy beneath Window.

C++ also allows multiple inheritance which permits a class to inherit attributes and operations from more than one class, thus providing more flexibility.

2.1.4 Polymorphism and Late Binding

Another powerful feature of object oriented programming is "Polymorphism", which is the multiplicity of implementations for a single method. The need for polymorphism arises when operations have more than one method. A technique called late binding, which means the procedure address is not bound to the procedure until the procedure call is made, is used to implement this feature and the programmer makes use of this by declaring a member function *virtual*. Thus a descending SubClass may redefine operations to suit its need. This is extremely important in cases where there are many subclasses from a class and the operation needs to be redefined for each of them. For example consider the SubClasses Frame, Scrollbar, Tty, Icon etc inheriting attributes and operations from a higher class Window, the operation redraw would be different for each subclass so the SuperClass Window should declare the operation redraw as *virtual*.

2.2 Reuse and Extensibility of Code

Creating new SubClasses of previously defined classes and using their previously implemented operations is called "Reuse". Generalization and Inheritance make code reuse possible. The combination of Inheritance and Polymorphism gives the user of object oriented code the remarkable benefit of being able to extend that code without having the source code. This is possible because inheritance operates across compile time boundaries. As long as the programmers have a description of the interface to a class (namely the header files), They can define a derived class

that inherits everything the base class has. The programmer can even selectively overload (because late binding occurs at run time), or redefine the behavior of a selected function to suit their needs. This means object libraries can be distributed without having to reveal algorithms. Also, an application using an object library is not necessarily limited by the specifications of objects in that library. If one or more objects in the library don't meet a programmer's needs, those objects can be modified by extending them as new classes.

2.3 X Window System Concepts

The X Window System, is a network based graphics windowing system that was developed at MIT. It has been adopted as an industry standard windowing system and is supported by many industry leaders. One of its major features is that it's not operating system dependent unlike most windowing systems, but is instead comprised entirely of "user-level" programs. It is based on what is known as a "client-server" architecture. The system is divided into two distinct parts: the "display server" and client. The display server provides display capabilities to many clients, handles user input (which could be through keyboard or mouse) and passes it to the clients. The clients are application programs that perform specific tasks. The local hardware is controlled by the server.

What has made X a standard is the fact that it is based on a network protocol called X-Protocol and which does not use system specific calls. X Protocol is a predefined set of requests, queries, replies and can be implemented on different computer architectures and operating systems, making X device independent. Another advantage of network based windowing system is that programs can run on one architecture while displaying on another. The display server controls all the resources of the client. A client needs to send a request to Xserver in X Protocol in order to do an operation. For example a client like a drawing program sends a request to draw a line on a local display in X Protocol. Many times the message may be a query from client.

2.3.1 Window Manager

Window Manager is a special client that manages the position and sizes of main windows of application on a server's display. It allows the user to move and resize windows, rearrange the order of windows in the window stack, create additional windows, convert windows into icons, etc. X consortium's "Inter Client Communications Conventions Manual (ICCCM)" defines the standards for interaction between window managers and other clients.

The window manager distributed by MIT is twm. There are two major industry enhancements, mwm (Motif window manager) and olwm(OPENLOOK window manager).

2.3.2 Writing X Application and User Interface Standards

To write an X application requires that it should receive X protocol messages. The lowest level of interface to X Protocol is a C interface named Xlib provided by MIT. Xlib provides complete interface to X Protocol by translating C data structures and procedures into X Protocol Events. However this interface is more extensive than required for most applications. Furthermore it becomes difficult to modularize the common functions and set up standards. Hence there are many toolkits available on top of Xlib. These toolkits handles basic things for the programmer and thus setting standards on user interface. There are two major standard graphical user interfaces used, namely OPENLOOK and Motif. Our choice was OPENLOOK, and the toolkit chosen was XView. Figure 2.3 illustrates the architecture of PNT.

2.3.3 XView toolkit and notifier based approach

XView is an object oriented toolkit in which each piece of user interface is an object from a particular hierarchy with a list of attributes. These attributes can be queried or set by message passing functions. XView objects have callback functions which are triggered by events. Callback are the functions to which the notifier passes control when the respective object is selected. These objects are represented in a class hierarchy and all objects are

SubClasses of an object called "Generic Object," see Fig 2.2. To pass messages to these objects a handle is returned by XView each time an Object is created.

XView is a notification based system, in which there is an object called notifier with whom control resides. All objects register their call back routines with the notifier and then pass control to the notifier. The notifier reads all the events, translates them to appropriate XView events and passes control to the object which has registered the callback for that event.

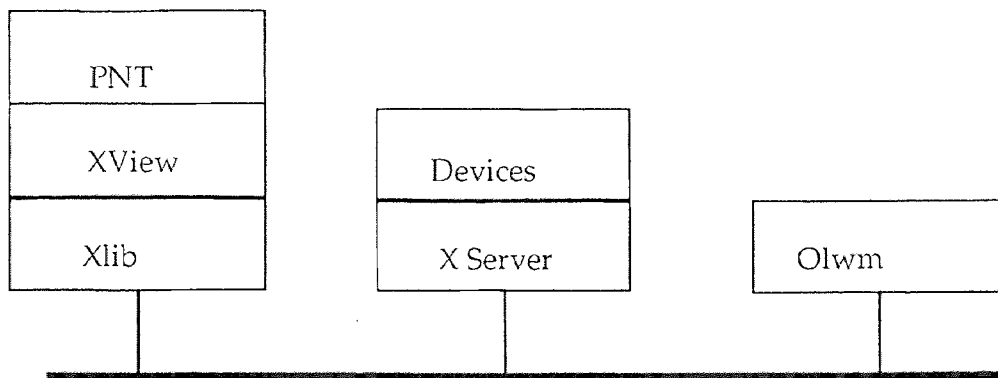


Figure 2.3 The Software Architecture of PNT

CHAPTER 3

DESIGN OF PETRI NET TOOL

3.1 How OOP is Applied to Design of PNT

The biggest challenge in object oriented programming is to select and define the objects relevant to the problem domain. The PNT has two major components, namely editor and simulator. The editor is developed using the object oriented toolkit XView which defines various user interfaces items as objects in a hierarchy and it contains an object namely "Notifier" with which the control resides and which distributes the events to various objects. These objects register their callback routine with the notifier. Thus all the Editor objects were standard[6].

The choice of simulator objects was inspired by the mathematical definition of Petri nets, a tuple of place, transition and arcs. There is a *basic_object* which contains the basic properties of both places and transitions, Object *transition* and *place* are both subclasses of *basic_object* and there is an object *arc* which contains object *segment_array* which is a collection of objects *segment*. This is so because in this model arcs are represented as a collection of segments. This model of arcs is a reasonable compromise between a straight line and a free form bitmap for the current application. There is an object *net* which contains the whole topology and is thus available to both the editor and the simulator. For simulation control there is an object called *controller*. The functionality is completed distributed. The various operations performed by an object decide its role.

The control flow is based upon message passing between objects. The picture will be clearer once the details of each object described later are understood. At start up all the necessary editor objects are created and each object registers its callback with the notifier so that the notifier should can control to it once the user selects an editor object by clicking a button. Also a controller object and a net object with nothing in it are created and control is passed to the notifier. As the user

draws or edits the net the editor object (the button which user clicked to say draw a place) tells the net object to add a place with a specific location on the canvas. This message goes through the controller. The net object has an operation to add a place which creates an instance of object place and sets its initial attributes. The net also updates its topology. When user clicks the step button to execute the net, control is passed to that instance of button through the notifier. The button tells the controller to step. It in turn queries each transition in the net to form a list of ready transitions, and then resolves the conflict among the ready transition, if any. Then the controller tells the transitions to fire and the actual fire operations are carried out by the transition objects. The following sections provide an in depth discussion of each object.

3.2 The Controller Object

The controller is a key simulator object. Its various attributes and operations are listed in Table 3.1. Attribute *steps* specifies the steps of simulation which could be set by the user through the interface. Attribute *rdy_trans* is the number of ready transitions and is used by its member functions for conflict resolution and firing. *list_rdy* and *size_list* are attributes for the list of ready transitions. The controller at run time forms a list of ready transitions. This list is implemented as a stretchable array which is initially created with a fixed size and is stretched if needed through the member function *Stretch_Array*. Another important attribute of controller is *net_handle* which provides the address of net object to both the controller itself at run time and to all the editor objects. At run time the sequence is *Check_Condition*, *Form_List*, *Resolve_Conflict* and *Execute*. The first checks if the ending condition set by user has been reached. The second forms the list of ready transition by querying each transition if its enabled. The third resolves conflicts among the ready transitions and the fourth actually fires the transitions. *Destroy_All* is used to destroy all the dynamically created Objects when the user either clears the net or quits the tool.

Table 3.1 Attributes and Operations of Controller Object

| Attributes | Operations |
|-------------|----------------------------|
| steps, mode | Destroy_All, Stretch_Array |
| rdy_trans | Check_Condition, Form_List |
| size_list | Resolve_Conflict, Execute |
| net_handle | Get_Net_Handle |
| list_rdy | Is_Conflict |

3.3 The Net Object

Net object contains the topology information about the Petri net on the canvas. Its attributes include the number of places, transition, input and output arcs in the net. It also holds the default identification number to be assigned to a new place or transition addition to the net (these numbers are internal and are used for net manipulation). It contains two stretchable arrays. One holds pointers to various places and one holds pointers to various transitions. Arc handles are kept with the transitions. The index of each place and transition pointer in the arrays is an internal counter which makes locating a specific place or a transition very easy. Another net attribute is its file name which can be changed by the user.

Major operations include the addition of places, transitions and arcs to the net. All user drawing operations lead to a message to the net object to add the respective component. Note that arcs added is also to the transition with which they are associated. Parallel to these operations are removal of place, transition and arc from the net. Operations *Get_File* and *Set_File* are used mainly when a user changes the file name or when a load/save are requested. *Get_Place*, *Get_Transition* are used for getting the handle to place and transitions and *Get_Pl_No*, *Get_Trans_No* are used internally to locate the internal index of places and transitions. Another operation is *Get_Arc* which is used when a user selects an arc to modify its tag or to delete it. Table 3.2 lists various attributes and operations of

class Net.

Table 3.2 Attributes and Operations of Class Net

| Attributes | Operations |
|--------------------------|------------------------------|
| file_name | Add Place, Transition, Arc |
| place_num, place_no | Remove Place, Trans, Arc |
| transition_num, trans_no | Get, Set File Name |
| input_num, output_num | Get Place, Transition, Array |
| size of arrays | Get Place/Transition No |

3.3 The Transition Object

The transition object is a subclass of object `basic_object`, so it inherits all the attributes and operations of class `basic_object`, described later. Although the net object has the complete topology, transition has enough information to decide whether it's enabled or not and also to fire. Thus by locally keeping some topology, significant computing time is saved during simulation. This is done by keeping arrays of pointers to arcs connected with each transition. The index of a pointer in the array is the internal place number to which the arc is connected. Other attributes of object transition are the sizes of these arrays and the numbers of input and output arcs. The state of transition, its orientation and priority are also its attributes.

The major operations are *Is_Enabled* and *Fire*. With pointers to arcs and knowledge about the places connected to the arcs, these operations are easily implemented. Other operations are addition and deletion of input and output arcs (actually called by net's add arc operation). There are operations to set or query the state, priority and orientation. The state is controlled by the controller object. The priority and orientation can be changed by the user. The usual operation of *Stretch_Array* is here because the array of pointers to arcs is stretchable. Table 3.3

lists the attributes and operations of transition object.

Table 3.3 Attributes and Operations of Transition Object

| Attributes | Operations |
|---------------------------|---------------------|
| state | Get/Set state |
| priority, orientation | Get/Set priority |
| array of pointers to arcs | Get/Set Orientation |
| size of arrays | Add/Remove arcs |
| number of connected arcs | Is_Enabled, Fire |

3.4 The Place Object

The place object is a subclass of `basic_object`, so it inherits all the attributes and operations of `basic_object`. Additionally it has attributes the `no_of_tokens` and `breakpt` which represent currently held tokens and execution end condition specified by user (defaults to don't care) respectively. No more information needs to reside with this object. It's main operations are `Get/Set_Tokens` and `Add/Remove_Tokens`. The first is normally used when the user queries or sets the number of tokens through the tag panel. The second is used during net execution. Table 3.4 describes its attributes and operations.

Table 3.4 Attributes and Operations of Place Object

| Attributes | Operations |
|---------------------------|-------------------|
| <code>no_of_tokens</code> | Get/Set Tokens |
| <code>breakpt</code> | Add/Remove Tokens |

3.5 The Arc Object

The object arc has among its attributes, *type* which could be either input, output or inhibit. *weight* tells the multiplicity of arcs between a place and transition of a given type. It also has the number of the place and the place handle to ease addressing for firing. It holds a pointer to object segment array, which actually contains all the segments comprising the arc.

The operations of object arc are *Get/Set_Type* which is set once and then can only be queried; *Get/Set_Place*, which is set initially and queried later; *Get/Set_Weight* which can be changed by the user at any time using the tag panel; *Get/Set_Handle*, which is set initially and queried for firing by the transition object and *Get/Set_Segment_Array*, used at creation and destruction time respectively. Table 3.5 lists all the attributes and operations of arc object.

Table 3.5 Attributes and Operations of Arc Object

| Attributes | Operations |
|----------------------|-----------------------|
| type | Get/Set Type |
| place_num | Get/Set Place |
| place_handle | Get/Set Weight |
| weight | Get/Set Place Handle |
| segment_array_handle | Get/Set Segment Array |

3.6 The Segment Array Object

The segment array is an array of straight line segments which comprise an arc. Its main attributes are *no_of_segments* and the stretchable array of segment pointers. Major operations are *Add/Remove_Segment* to array and *Stretch_Array*. This object is created when a user starts drawing an arc. Each segment is added until the completion of arc. Then the Object segment array is added to arc object. While editing an arc *del_segment* removes the last segment from the array. Table 3.6 lists

all the attributes and operations

Table 3.6 Attributes and Operations of Segment Array Object

| Attributes | Operations |
|-------------------|--------------------|
| array of segments | Add/Remove Segment |
| size of array | Stretch_Array |

3.7 The Segment Object

Object segment is nothing but a straight line whose attributes are the coordinates of its endpoints. Its operations are *Get/Set_Coordinates* which are used at creation and destruction time. Table 3.7 lists the attribute and operation of class segment.

Table 3.7 Attributes and Operations of Segment Object

| Attributes | Operations |
|------------|-----------------|
| x1, y1 | Get_Coordinate |
| x2, y2 | Set_Coordinates |

3.8 The Basic_Object

basic_object is a superclass of both transition and place. It's attributes are number (this is the user defined number), coordinates on the canvas, and comment (upto 50 chars). The various operations are *Get/Set_Number*, *Get/Set_Comment* and *Get/Set_Coordinates*. The first are triggered by the user through the tag panel. The coordinates can be queried. They cannot be altered unless the place/transition is deleted and redrawn. Table 3.8 lists all the attributes and operations of *basic_object*.

Table 3.8 Attributes and Operations of Basic_Object

| Attributes | Operations |
|------------|---------------------|
| number | Get/Set_Number |
| comment | Get/Set_Comment |
| x, y | Get/Set_Coordinates |

CHAPTER 4

PNT USERS MANUAL

4.1 Overview of PNT

PNT has an easy to use graphical user interface. First draw a Petri net representation of the system to be modeled. Then the net is executed after setting various conditions to control the simulation run. The tool runs under OpenWindows environment, so it is necessary to run OpenWindows before running PNT. To run the tool type "pnt" from a command tool.

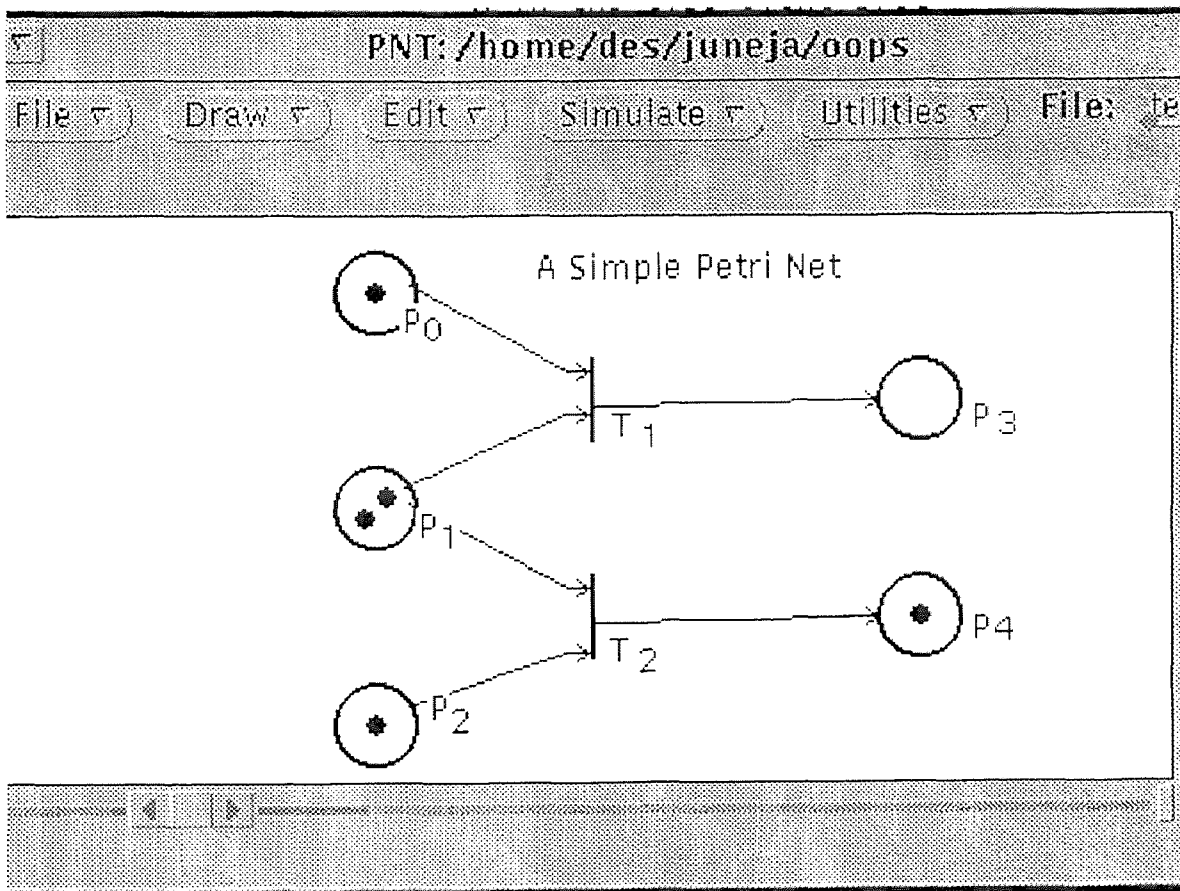


Figure 4.1 Petri Net Simulator in OpenWindows Canvas

The header of the tool consists of "PNT:" followed by the current working

directory, see Fig. 4.1. Below the header is the control panel where there are five menu buttons, namely File, Draw, Edit, Simulate and Utilities. Each of these buttons has a down arrow which indicates that they are pull down menus (the other kind is pull right). For the sake of simplicity the following abbreviations are used in this chapter, LMB for left mouse button and RMB for right mouse button. If the RMB is clicked a menu in the downward direction pops up. The control panel also has a File name Below the control panel is the drawing canvas where the Petri net is created. The whole frame can be resized and the canvas has horizontal and vertical scrollbars.

The File menu is used for loading and saving the net to a file on disk and for exiting the tool. The Draw menu is used for drawing the net on the canvas, it has all the entities which constitute a net namely place, transition(horizontal and vertical), arc(normal and inhibit) and tokens. The Edit menu has various options for editing the drawing of the net and also changing the attributes of the various entities. The Simulate menu, as the name suggests, is used for simulating execution. Various conditions for the simulation run can be set. Utilities provide useful information which includes an execution log and a verify matrix of the net drawn.

It is important to remember that in OPENLOOK interface one must press the RMB on the menu button and should hold the RMB down while scrolling through various choices and should release the RMB when the desired choice is selected. To acquire basic familiarity with OpenWindows environment it is strongly suggested to use the online tutorial for OpenWindows. The following sections discuss the functionality of each menu and button in detail.

4.2 The File Menu

The file menu is typically used in the starting and end of a session. When the tool is started, the default File name is noname which can be changed by typing in the "File:" item on the control panel. Fig 4.2 shows the File menu

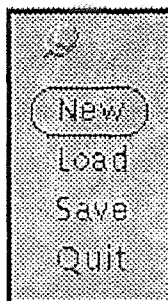


Figure 4.2 The File Menu in OpenWindows Canvas

4.2.1 The New Button

To start a session from scratch, select the New option of the File menu. If there is currently an unsaved net on the canvas, a warning will be given to save it first. The new session will use the file name from the panel as the filename for saving net. If the file already exists then a warning will be given that the file will be overwritten. The New button should be typically used when a net simulation has finished and another one is to be started from scratch.

4.2.2 The Load Button

To load a net previously saved to a file, select the Load option after specifying the name of the file on the control panel. Loading a net will remove the current net on the canvas, so if the current net on the canvas is not saved, PNT will warn to save the net first. If an illegal file name is specified PNT will prompt to input the correct filename. The file name should be from the current working directory or it should include complete path.

4.2.3 The Save button

It is extremely important to save the work before quitting the tool or loading another net. The net under construction should be periodically saved. The editor keeps tracks of the status of current net, saved or not. If not it prompts to save before doing anything that will destroy the currently displayed net.

4.2.4 The Quit Button

To exit PNT use this button and not the OpenWindows quit window button. Although both will ask for confirmation before closing the window, only PNT quit prompts for saving an unsaved net.

4.3 The Draw Menu

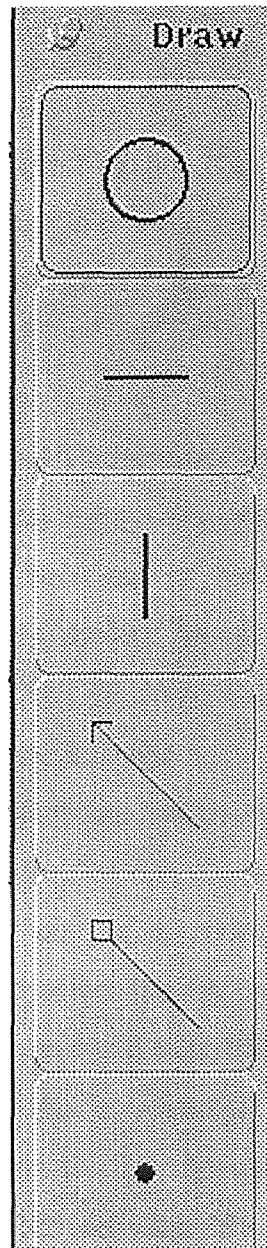


Figure 4.3 The Draw Menu in OpenWindows Canvas

Once the system to be analyzed in terms of a Petri net has been modelled, the Draw menu provides all the options to draw the net on canvas of the tool. When RMB is clicked on the Draw button of the control panel it shows the five choices of Fig. 4.3 namely place, horizontal transition, vertical transition, normal arc, inhibit arc and token. The selection button has an image of these entities rather than text. An important feature of the Draw menu is that once an entity is selected then as many instances of that entity can be drawn as desired.

4.3.1 The Place Button

To draw a place on the canvas, select the place(circle) option from the Draw menu. A cursor move and LMB click at the desired location deposits a new place on the canvas as many times as desired. To exit this mode select another desired menu option.

4.3.2 The Horizontal and Vertical Transition Buttons

Both transitions have the identical logic functionality. To draw a transition on the net first select one of the bars (horizontal or vertical) from the Draw menu. Now move the mouse arrowhead to the desired location on the net and click the LMB.

4.3.3 The Normal and Inhibit Arc Buttons

These buttons have identical interfaces but provide different logic functionality. As mentioned before, arcs are implemented as a collection of segments in this model. The interface to draw an arc is more complicated than other draw operations. To draw an arc first select the arc (normal or inhibit) from the Draw menu. Drawing of arc is done from head to tail. First click the LMB on the place or transition where the arc ends. An arrowhead should appear at that place or transition. Sometimes it is needed to click more than once as it is possible that the cursor wasn't in the invisible tile including the place or transition. If there are already many arrowheads on the selected place or transition it is possible that arrowheads might overlap, but this does not affect the functionality. The

arrowhead will have different appearances for normal and inhibit arcs. Once the arrowhead appears only the RMB should be used to specify segment endpoints. One can draw an arc as a chain of many segments (minimum being the arrow head and one segment). During this process one can draw arcs crossing places, transitions and other arcs, if necessary. The process will end when the RMB is clicked at a place or transition.

If an attempt to draw an illegal arc like an arc between two places or an arc between two transitions, PNT generates a warning and deletes the arc itself. When an appropriate end point of an arc is selected, a notice is put up to announce that drawing of arc is complete. Use the Tag button from the edit menu to change the weight of the arc. There is a button in the edit menu called DelSeg which deletes the last segment drawn. It is extremely useful for editing while in middle of constructing an arc. PNT does not permit to draw multiple arcs of the same type between a given place/transition pair, i.e. for a given place and transition the user can only draw one input(or inhibit) and one output arc.

4.3.2 The Token Button

This button allows addition of tokens to a place. First select token (dot) from the Draw menu and then move the mouse arrowhead to the desired place on the net and click the LMB. If the click is outside a place, a notice is put up saying click on a place. The visual appearance of tokens is dots for up to four tokens in a place and then numbers for more.

4.4 The Edit Menu

This menu is used for a number of modifications that can be done to a net structure and to certain attributes of net entities. The various choices in this menu are Tag, ArcTag, Eraser, EraseText, DelArc, DelSeg, Clear and KillText. See Fig. 4.4. Tag is used to query the attributes of a given place, transition or arc. It can also change certain of their attributes. Eraser is used to remove places, transition and tokens. EraseText is used for removing text from the canvas. For removing arcs there is a

separate button DelArc because selecting an arc is different than selecting any other net entity. DelSeg is used while drawing an arc to remove the last drawn segment. It is possible to remove an entire arc by continuously using it. Clear clears the complete canvas and destroys the net in memory. KillText clears all the text from the canvas without effecting the net.

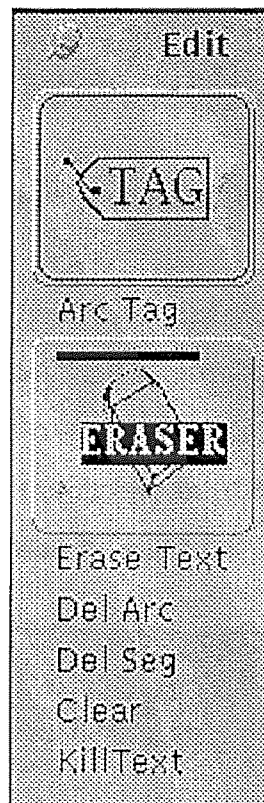


Figure 4.4 The Edit Menu in OpenWindows Canvas

4.4.1 The Tag and ArcTag Button

Tag provides access to entity parameters. It can be used with a place, transition or an arc. To use it one selects the Tag button from edit menu and goes to the desired place, transition or arc. To select a place or a transition click the LMB over the place or transition on canvas. For an arc first select ArcTag button and then click the LMB on its head and RMB on its tail. Once an entity is properly selected a panel pops up describing the attributes of that object. Tag panel has two buttons Apply

and Cancel, Use apply to change attributes and Cancel to remove the Tag panel without changing its attributes. Figures 4.5, 4.6 and 4.7 show the tag panels for a place, transition and arc respectively.

Figure 4.5 Tag for a Place in OpenWindows Canvas

For the Place, its number, comment and stop marking can be changed. For the transition the number, priority and comment can be changed. For an arc the user can change the weight and comment. Once the changes have been made on the tag panel click the LMB on the Apply button to bring them into effect. Other attributes are read only and cannot be edited, for example type of an arc.

Figure 4.6 Tag for a Transition in OpenWindows Canvas

Figure 4.7 Tag for an Arc in OpenWindows Canvas

4.4.2 The Eraser and EraseText Buttons

The eraser can remove anything from a net but an arc or text. Eraser is like a loaded gun and must be used carefully. To erase select Eraser from the Edit menu. Then click the LMB over the place or transition to remove it. For a place or transition the implication of erasure is to remove all the arcs connected to that place or transitions and also the tokens for a place. To remove a token press RMB on the place from which tokens have to be removed. Tokens can be set also by the Tag panel for place. There will be a warning before removing both places and transition that "associated arcs will also be deleted." Tokens will be removed without any warnings. Use Eraser carefully.

EraseText is used for removing text from the canvas. To erase text first select EraseText from the Edit menu. Then select a rectangle in the canvas enclosing the starting point of the text strings by clicking LMB on its diagonal points. If the rectangle encloses the starting point of more than one text item, all of them are erased without any warning.

4.4.3 The Clear and KillText Buttons

Clear removes everything from the canvas and destroys all the dynamically allocated data structures. This means loss of information if net is not saved to a file on disk. If the current work is unsaved a warning is given by PNT to save the net. No warnings are given if the current work was saved.

Kill text will remove all the text from the entire canvas without any warning. Unlike Clear the net structure is not affected. This is useful when current text is stale but the net is still useful

4.4.4 The DelSeg and DelArc Buttons

These are used for editing arcs. DelSeg is used only when during drawing arc and is inoperable once an arc is complete. A notice is put up saying "no unfinished arc" if the button is clicked otherwise. DelSeg removes the last segment drawn and can be used successively until all the segments of the current arc are deleted. Once an

arc is completed, the only way it can be deleted is by DelArc. A separate button is provided for deleting arcs because selecting an arc on the canvas requires two clicks. To delete an arc first select its head with the LMB and then its tail with RMB. The order is important

4.5 The Simulate Menu

This menu should be used once a complete net has been drawn and verified or loaded from a file. It has three choices Step, Run and Breakpt. See Fig 4.8. Simulate in step mode by Step choice from the Simulate menu or simulate in run mode by selecting Run. For run mode stopping condition can be specified by selecting the Breakpt choice, see Fig. 4.9.

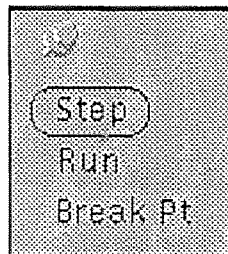


Figure 4.8 Simulate Menu in OpenWindows Canvas

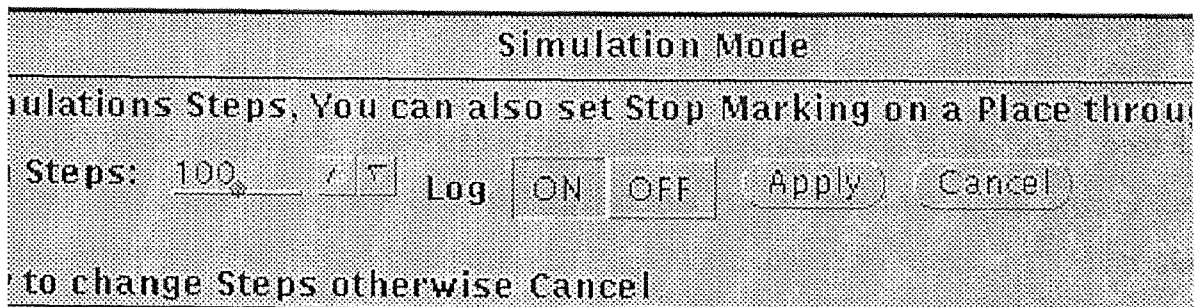


Figure 4.9 Panel for Simulation Mode in OpenWindows Canvas

Breakpt brings up a panel in which the maximum number of steps can be set and a toggle option to create an execution log to file on disk. See Fig. 4.9. The default limit is 100 steps, but it can be modified. Use the tag for place to specify stop

marking as another stop condition, see Fig 4.5. The default for this parameter is "don't care" meaning the marking of a place is ignored. If more than one place has a token stop condition, the simulation halts when any condition is met. In both run and step modes, if no transition in the net is enabled a notice saying "Deadlock, no enabled transitions" is displayed.

4.6 The Utilities Menu

It has five choices Verify, Type, Log, DelLog and Redraw. See Fig 4.10.

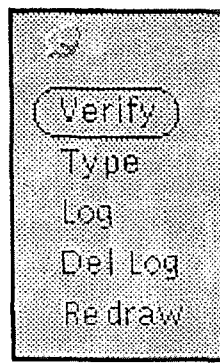


Figure 4.10 The Utilites Menu in OpenWindows Canvas

Verify provides a tabulated topology of the net and attributes of various net entities in the order - transitions, connected arcs and places. Fig. 4.11 shows a sample verify file.

Type provides the feature to write optional text on canvas from the keyboard. Text can be written anywhere on the canvas to mark places, transitions, arcs or for any other purpose. Select Type from Utilities menu and then click the LMB on the desired location on the canvas and start typing. Press the keys control and s together to delete the last character. During one selection text can be written only in a straight line. To place text elsewhere select Type again and click LMB at the new location. See Fig. 4.1 for examples of superimposed text on net.

Log is a detailed description of the simulation which is also saved to a disk file. It shows which transitions were enabled, which of them fired after conflict

```

-----Verify File-----
Quit )
-----Net Information-----
No of Places = 10           No of Transitions = 10
  No of Input arcs = 12       No of Output Arcs = 12
-----Transition Information-----
Transition no = 0, state = 0, priority = 1, or = v
  input arc = 1, output arcs = 1
-----Connected Arcs-----
Arc Type = 1, pl_no = 0, wt = 1, seg_no = 4
Arc Type = 0, pl_no = 1, wt = 1, seg_no = 4
-----Transition Information-----
Transition no = 1, state = 0, priority = 1, or = v
  input arc = 1, output arcs = 1
-----Connected Arcs-----
Arc Type = 1, pl_no = 0, wt = 1, seg_no = 4
Arc Type = 0, pl_no = 2, wt = 1, seg_no = 4
-----Transition Information-----
Transition no = 2, state = 0, priority = 1, or = v
  input arc = 2, output arcs = 1
-----Connected Arcs-----
Arc Type = 1, pl_no = 1, wt = 1, seg_no = 4
Arc Type = 1, pl_no = 3, wt = 1, seg_no = 4
Arc Type = 0, pl_no = 4, wt = 1, seg_no = 4
-----Transition Information-----
Transition no = 3, state = 0, priority = 1, or = v
Use Quit to exit this window

```

Figure 4.11 Sample Verify Window in OpenWindows Canvas

resolution, and how the conflict was resolved (priority difference or random). The markings before and after firing are also displayed. Fig. 4.12 shows a sample from a log file.

There is a button DelLog to delete the old log file. Otherwise the log of a current run is appended to the existing file. Since the log file may become very large, it is recommended to be periodically deleted. Redraw will redraw the net and the text on the canvas using the information from the net object and text object. This is very useful after screen refresh from Olwm, and after editing net.

```

-----
Log File
-----
Quit
-----
-----Marking of Net is -----
Place No  0  1  2  3  4  5  6  7  8  9
Tokens    4  0  0  1  0  0  0  0  0  0

PNT: Transition Number 0 ready
PNT: Transition Number 1 ready
PNT finds following transition In conflict
Transition Number  0  1
PNT: Transition Number 0 lost to 1 in random resolution
PNT: Step 0 of 100
PNT: Firing Transition Number 1
-----
-----Marking of Net is -----
Place No  0  1  2  3  4  5  6  7  8  9
Tokens    3  0  1  1  0  0  0  0  0  0

-----
-----Marking of Net is -----
Place No  0  1  2  3  4  5  6  7  8  9
Tokens    3  0  1  1  0  0  0  0  0  0

PNT: Transition Number 0 ready
PNT: Transition Number 1 ready
-----
Use Quit to exit this window
-----

```

Figure 4.12 Sample Log Window in OpenWindows Canvas

4.7 Control Flow in PNT

It is important to know the internal control flow in PNT in order to use the tool efficiently. At a particular instance PNT has a fixed state, out of a set of finite number of states. The various states the tool can be in are the five draw operations, namely place, horizontal transition, vertical transition, normal arc, inhibit arc and token. Besides draw the tool can be in erase, tag or delete arc state. Once one of buttons from this set is selected, the state of PNT corresponds to the button and

does not change until another button from this set is selected. Selecting a button not in this set, such as Step does not change the state of the tool. For example when the place draw button is selected, PNT enters the place draw state and as many places on the canvas as desired can be added. One can step and still draw the place on the net without selecting the place draw button again as the state of the tool has not changed. The buttons which do not affect the state of tool are all the buttons in File, Simulate, Utilities menu, Clear & DelSeg. These buttons do their respective job and give back control to the current state of PNT. This feature provides the user with the convenience of not needing to go to the menu options repeatedly. On the other hand ignorance of this feature could lead to unwanted operations.

CHAPTER 5

CONCLUSION

5.1 Portability and Enhancement Issues

Here is a brief outline on how to enhance the tool by adding features like time and color by using the inheritance and polymorphism features of object oriented programming. For simulating timed Petri nets the class controller should be subclassed to a new class, timed controller, which inherits everything from controller and adds the attribute clock and overloads the operation *execute*. Class net remains the same, but class transition needs to be subclassed into a new class timed transition which inherits everything from class transition and adds the attributes time and type. It overloads the operations *Is_Enabled* and *Fire* using the polymorphism feature of object oriented programming, and it should also add operations like *Time_Remaining*, *Get_Type*, *Set_Type*. For adding color the classes place and arc need to be subclassed with new attributes representing color.

Porting the application to a new hardware will require recompilation. The compiler used is AT&T's C++ compiler which is very much the standard. It is compatible to other C++ compilers. Even on PC's with C++ compilers like Borland C++ recompilation would not be a major problem. The user interface is designed using the XView toolkit. XView was developed by Sun Microsystems as a migration path from SunView, as SunView was hardware and operating system dependent and was not networkable. XView is implemented purely on top of X11 and during its implementation there were relatively few problems in porting the toolkit to other platforms. In fact, the XView toolkit is available from MIT's X Consortium and many ports to a variety of platforms are currently available. XView comes as a standard distribution with all Sun machines. The only other consideration is the user interface standard. There as is currently no clear consensus on that issue and we had no choice but OPENLOOK. If the world goes

for motif in the future, there are even now applications (like Molit) available to convert OPENLOOK applications into motif.

5.2 Summary

The object oriented paradigm was successfully applied to design of PNT. Object oriented programming gave greater meaning to design phase as first all the objects with their attributes and operations were selected and then the corresponding design was implemented. It also eased the task of debugging as each object was individually implemented and tested. Once this was done the whole application was integrated by defining message passing sequences between the objects to carry out a required task. XView toolkit proved very useful to comply with the user interface standard and its object oriented design served our objective of using object oriented design throughout.

APPENDIX

```

/*
*****
*   Copyright NJIT 1993 -- All rights reserved
*
*       Authored by Himanshu Juneja
*****
*
*/
#include "net.h"
class controller {
int mode; // simulation mode
net *net_handle; // pointer to the current net
int size_list; //size of array;
transition **list_rdy; // array of ready transitions

public:
controller(char *);
int rdy_trans; // no of rdy transitions

void Stretch_Array(void);
void Destroy_All(void);
net *Get_Net_Handle(void);
void Form_List(void);
void Resolve_Conflict(void);
int Check_Condition(void);
void Execute(void);
int Save_Net(char *);
int Load_Net(char *);
};

```

File A.1 Class definition for Controller

```

/*
*****
*   Copyright NJIT 1993 -- All rights reserved
*
*       Authored by Himanshu Juneja
*****
*
*/
#include "transition.h"
const int PLACE = 1;
const int TRANSITION = 2;
const int ERROR_A = -2;
const int ERROR_B = -3;
const int ERROR_C = -4;

```

```

class net {
public:
int place_no,transition_no,input_no,output_no;
int place_num, trans_num;
char file_name[100];
int size_place,size_transition;
transition **trans_list;
place **place_list;

net(char *);
int Add_Place(int,int);
int Add_Transition(char,int,int,int);
int Add_Arc(int,int,int,seg_array*);
int Remove_Place(int);
int Remove_Transition(int);
int Remove_Arc(int,int,int);
char *Get_File(void);
int Set_File(char *);
void Stretch_Array(int);
place *Get_Place(int,int);
place *Get_User_Pl_No(int);
int Get_Pl_No(int,int);
transition *Get_Transition(int,int);
int Get_Trans_No(int,int);
arc *Get_Arc(int,int,int);
};

```

File A.2 Class definition for net

```

/*
*****
* Copyright NJIT 1993 -- All rights reserved
*
* Authored by Himanshu Juneja
*****
*/
#include "arc.h"

const int READY = 1;
const int NOT_READY = 0;

class transition : public basic_object {
// attributes for class transition

int state; // state of transition ready/not ready

```

```

int priority; // priority of transition
char orientation; // horizontal/vertical

public:
int input_no, output_no; // number of input and output arc
int size_input, size_output; // size of arrays of pointers to
input&output arcs
arc **input; // arrays of pointers to input & output arcs
arc **output;
transition(char, int, int, int, int); // constructor for class
transition
char Get_Orientation(void); // returns the orientation
void Set_Orientation(char); // sets the orientation
int Get_State(void); // returns the state of transition
void Set_State(int); // sets the state of transition
int Get_Priority(void); // returns the priority of transition
void Set_Priority(int); // sets the priority of transition
void Stretch_Array(int); // stretches the input/output array
int Add_Arc(int, int, arc *); // adds an arc to the input/output
array
void Remove_Arc(int, int); // removes an arc from the input/
output array
int Is_Enabled(void); // returns whether the transition is
enabled or not
void Fire(void); // fires the transition
};

```

File A.3 Class definition for transition

```

/*
*****
* Copyright NJIT 1993 -- All rights reserved
*
* Authored by Himanshu Juneja
*****
*/
#include "basic_object.h"

const int SUCCESS = 0;
const int FAIL = -1;

class place : public basic_object {
// attributes of class place
int no_of_tokens; // no of tokens in place

public:

```

```

place();
place(int,int,int); // constructor for class place

void Add_Tokens(int);
int Remove_Tokens(int);
int Get_Tokens(void);
void Set_Tokens(int);
};

```

File A.4 Class definition for place

```

/*
*****
* Copyright NJIT 1993 -- All rights reserved
*
* Authored by Himanshu Juneja
*****
*/
#include "place.h"
#include "seg_array.h"

const int INHIBIT = 2;
const int INPUT = 1;
const int OUTPUT = 0;

class arc {
// attribute of class arc
int type; // type of arc input/output
int place_no, transition_no; // no of associated place and
transition
int wt; // weight of arc
place *place_handle; // pointer to the associated place
seg_array *seg_handle;

public:
arc(int,int,int,place *,seg_array *); // constructor with
intialization

int Get_Type(void); // returns type of arc
void Set_Type(int); // sets type of arc
int Get_Place(void); // returns associated place no
void Set_Place(int,place *); // sets place no
int Get_Transition(void); // returns associated transition no
void Set_Transition(int); // set transition ni
int Get_Wt(void); //returns wt of arc
void Set_Wt(int); // sets wt of arc

```

```

place *Get_Handle(void); // returns the place handle
seg_array *Get_Seg_Array(void); // returns pointer to segment
array
void Set_Seg_Array(seg_array *); // sets the current segment
array
};

```

File A.5 Class definition for arc

```

/*
*****
* Copyright NJIT 1993 -- All rights reserved
*
* Authored by Himanshu Juneja
*****
*/
#include "segment.h"
#include <stdio.h>

class seg_array {
int size_array;

public:
int no_of_segments;
segment **s_array; // array of pointers to segment
seg_array();

void Stretch_Array(void);
void Add_Segment(int, int, int, int);
void Remove_Segment(void);
};

```

File A.6 Class definition for segment array

```

/*
*****
* Copyright NJIT 1993 -- All rights reserved
*
* Authored by Himanshu Juneja
*****
*/

class segment {
public:
int x1;
int y1;
};

```



```

int x2;
int y2;

segment(int, int, int, int); // constructor for class segment

);

```

File A.7 Class definition for segment

```

/*
*****
* Copyright NJIT 1993 -- All rights reserved
*
*       Authored by Himanshu Juneja
*****
*/
#include <iostream.h>
#include <stdio.h>
#include <string.h>

class basic_object {
// attribute for the basic object
int no; // the number assigned to a place/transition
int loc_x, loc_y; // the x,y coordinates of the object on the
canvas
char comment[50];

// member functions are public
public:

//constructor and destructor
basic_object();
~basic_object();

// member functions are public
int Get_No(void);
void Set_No(int);
int Getloc_x(void);
void Setloc_x(int);
int Getloc_y(void);
void Setloc_y(int);
char *Get_Comment();
void Set_Comment(char *);
};

```

File A.8 Class definition for basic_object

REFERENCES

1. Chen, Yan "Timed Petri Net Simulation of Flexible Manufacturing Systems." *Master's thesis, Department of Electrical and Computer Engineering, New Jersey Institute of Technology, December (1992).*
2. Chiola "A Graphical Petri Net Tool for Performance Analysis" *Workshop on Modeling Techniques and performance evaluation France, March (1987).*
3. Desai, Sanjay "A Graphical Tool for the Simulation of Colored Petri Nets." *Master's thesis, Department of Electrical and Computer Engineering, New Jersey Institute of Technology, December (1991).*
4. Dicesare, Frank and Alan A. Desrochers. Modeling, Control, And Performance Analysis of Automated Manufacturing Systems. *Control and Dynamic Systems, Vol. 47, Academic Press Inc. (1991).*
5. Feldbrugge, F., "Petri Net Tools," Philips Data Systems, Netherlands. (1988).
6. Haller, Dan XView Programming Manual, an OPENLOOK Toolkit . *O'Reilly & Associates Inc, California, (1991).*
7. Gilanli, Arsalan. "A Graphical Editor for Petri Nets." *Master's Thesis, Electrical Engineering Department, New Jersey Institute of Technology, (1989).*
8. Murata, Tadao. "Petri Nets: Properties, Analysis and Applications." *Proceedings of The IEEE, Vol. 77, No. 4, April (1989): 541-579.*
9. Peterson, James L. Petri Net Theory and the Modeling of Systems. *Prentice-Hall Inc. Englewood Cliffs, New Jersey (1981).*
10. Quercia, Valerie and O'Reilly, Tim X Window System User's Guide. *O'Reilly & Associates Inc., California (1991).*
11. Shukla, Ashish and Anthony Robbi. "A Petri Net Simulation Tool." *Proceedings of 1991 IEEE Int. Conference on Systems, Man, and Cybernetics, Vol. 1, No.1, October (1991): 361-366.*
12. Siddiqi, Javaid Aslam. "A Graphical Tool for the Simulation of Timed Petri Nets." *Master's thesis, Department of Electrical and Computer Engineering, New Jersey Institute of Technology, December (1991).*

References continued

13. Stroutsrop, Bjourne The C++ Programming Language .
Addison and Wesley Publishing House, (1991).
14. Zhou, MengChu, Kevin McDermott, Paresh A. Patel and Tenian Tang.
“Construction of Petri Net Based Mathematical Models of an FMS Cell.” *Proceedings of 1991 IEEE Int. Conference on Systems, Man, and Cybernetics, Vol. 1, No.1, October* (1991): 367-372.
15. Zhou, MengChu and Dicesare, Frank. Petri Net Synthesis of Discrete Event Control of Manufacturing Systems. *Kluwer Academic Press* 1993.